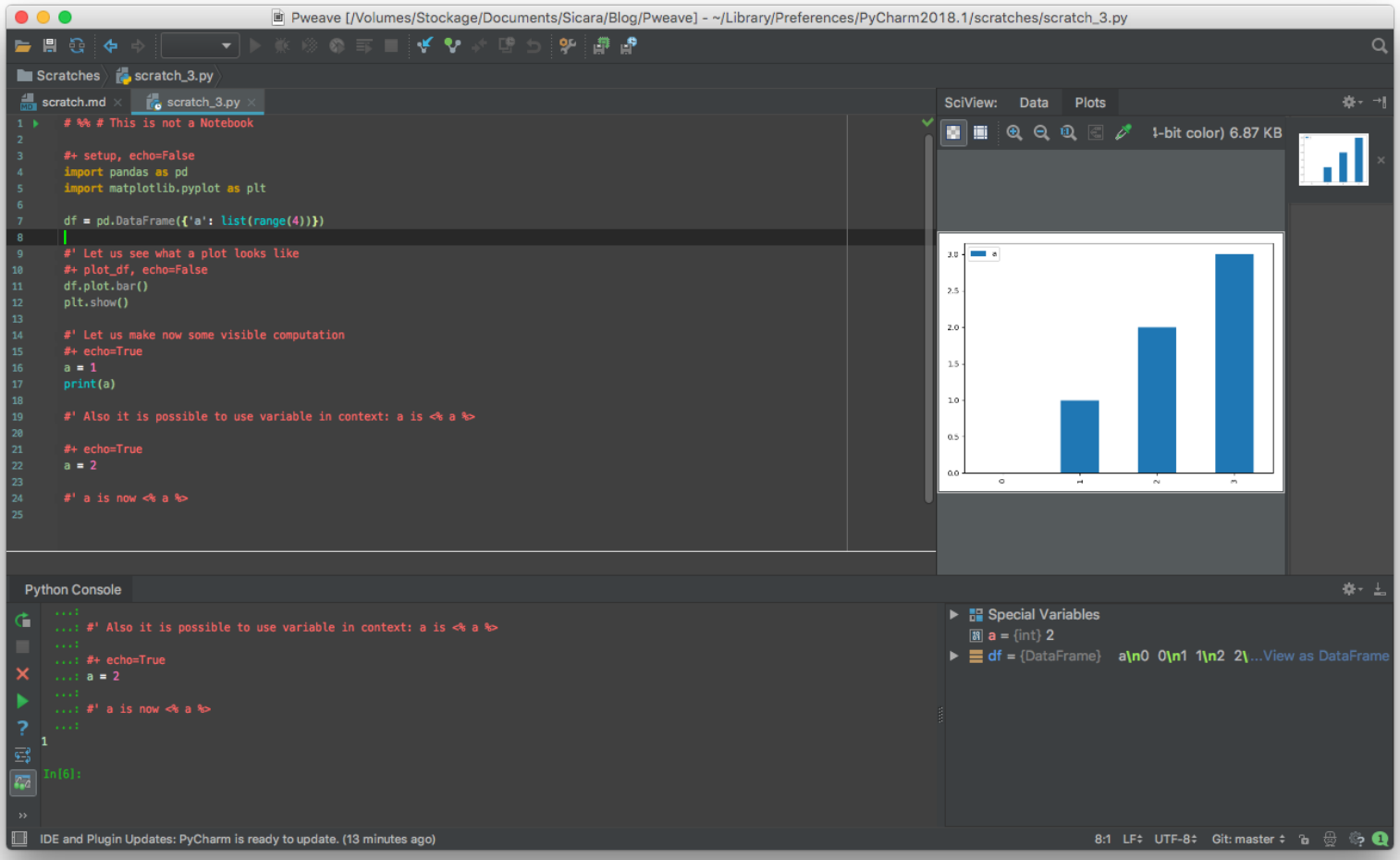
Ideal Notebook

From notebook prototyping to production the right way

Clément Walter Follow
Feb 25 · 5 min read



Jupyter notebook has been reported as the preferred prototyping tool for data scientist. This post presents the fast pace from EDA to API. Without Jupyter.

. . .

Jupyter main features are:

- inline code execution
- easy idea structuring

- nice displays of pictures and dataframe

This overall flexibility has made it a preferred tool compared to the more rustic iPython command line. However it should not be forgotten that this is not more than an REPL where you can navigate efficiently throughout the history. Thus it is not a production tool.

However, tons of machine learning developers have experienced the deep pain of refactoring a deep learning notebook into a real algorithm in production (also reddit or stackoverflow).

Keeping a lean mindset, we should strive to reduce waste as much as possible.

Introduction

At Sicara, we build machine learning based products for our customers:

- machine learning: the customer comes with a business need and we have to deliver a satisfying algorithm as fast as possible;
- we build products: we need to develop in a production-ready mindset. Algorithms are deployed in the cloud, served and updated with APIs, etc.

First of all you definitely need a versioning tool which is a pain with Jupyter (also reddit, reddit again, quora). Not only for your code, but also for your experiments. You need to be able to re-run any results got so far with 100% confidence. How often come data scientists with results they cannot reproduce?

Furthermore, when using notebooks, people often tend to mix three kinds of usage:

1. development: defining methods and tools to actually do something;
2. debugging/applying: running the piece of code with real data to see what is going on;
3. visualization: presenting the results in a clean and reproducible output.

In order to reduce waste, these steps should be clearly defined and separated so as to be able to change one without the other and vice versa. I have come to the conclusion that

- to produce high-quality tested code, better using a first-class IDE
- to debug code, there are visual debugging tools

- to write down reports, I am more comfortable with an expressive markup language (markdown, reST, Latex)

Fortunately a well-configured IDE can do all of these things. For instance if you come from the R community you certainly use RStudio which allows you to do so:

- native code completion, auto-fix, etc.
- direct visual debugging
- Rmarkdown/knitr/Sweave to generate dynamic and beautiful reports.

Develop production-ready code

As soon as you want to make an experiment, i.e. write a method to do something to your data, you should think about its usage, limit case, etc. Do it in a separate file, document and unit-test it. Doing so you make sure that:

- your method actually does what you want;
- your code can be safely used somewhere else in your project.

Because you will have to organize your tools, it makes you think about the structure of your pipeline, the things you need, what you are likely to change, etc. Python and R both allow for fast unit testing. Better spend 10 minutes writing down limit cases that 10 hours debugging wrong outputs!

For the sake of clarity, unit test are never in the same file as the one defining the method. But with Jupyter you end up doing so.

Debug and display

At this step, you have your new fully functional piece of code. Time to try it on *real data*! This is where notebooks can be found very convenient because of their cell mechanism. However it is a tool switch. Why would you quit your IDE with all you shortcuts and comfort to run code into your web browser? What you need is inline execution of your code directly into your IDE.

A tool like PyCharm has a native support of this feature: execute selected code or script with a single keyboard shortcut (action *Execute selection in console* or *Execute cell in console*). Furthermore its console runs the iPython ones with a very nice *Variables* tool window. In scientific mode you can also display and change plots and dataframes/arrays within the IDE.

Alternatively you can use tools like VSCode or Atom with Hydrogen for such features as well.

Report and share

At this point you should have your tested code in some directory in your project and a plain python file running it onto your data.

```
|-- project
     |-- notebooks
           |-- data_analysis.py
     |-- tests
           |-- do_something_test.py
     |-- utils
           |-- do_something.py
```

You have run it inline into your IDE and checked the results, they are great! You job is almost done: you need to report them to the team to justify the migration of the algorithm to your new version. Or maybe you are about to write a paper for the next NeurIPS conference.

You need to explain your logic and make a step-by-step clear explanation to prove your results. Of course you don't want to retype everything in another file; reporting is boring.

This is why there exists tools for literate programming. Documentation tools like Sphinx are built with that spirit: write your code and the documentation into the same file and generate a readable version from it.

For your python notebook, I recommend using Pweave. This is so far the best portage of knitr I have found. Not also that Rmarkdown fully supports python cells (or a mix of R and python).

In any case, I have found using the Pweave `pypublish` command is the most efficient. Just add comments to your scripts and run:

```
pypublish data_analysis.py
```

to generate a clear shareable html from it! Every commented lines are markdown interpreted, every *cell* (or code block) can be displayed or hidden, etc.

For instance running `pypublish` with this notebook (note the special comment tags `#'` , `#+` and `# %%`):

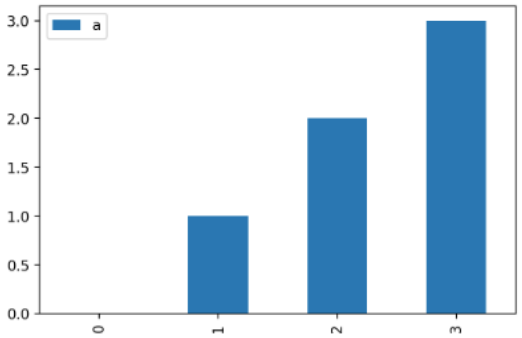
```
# %% # This is the title of the notebook


#+ setup, echo=False
import pandas as pd


df = pd.DataFrame({'a': list(range(4))})


#' Let us see what a plot looks like
#+ plot_df, echo=False
df.plot.bar()


#' Let us make now some visible computation
#+ echo=True
a = 1
print(a)


#' Also it is possible to use variable in context: a is
<% a %>


#+ echo=True
a = 2


#' a is now <% a %>
```

generates this report:

This is the title of the notebook

Let us see what a plot looks like

```
<matplotlib.axes._subplots.AxesSubplot at 0x1143755c0>
```



Let us make now some visible computation

```
a = 1
print(a)
```

```
1
```

Also it is possible to use variable in context: a is 1

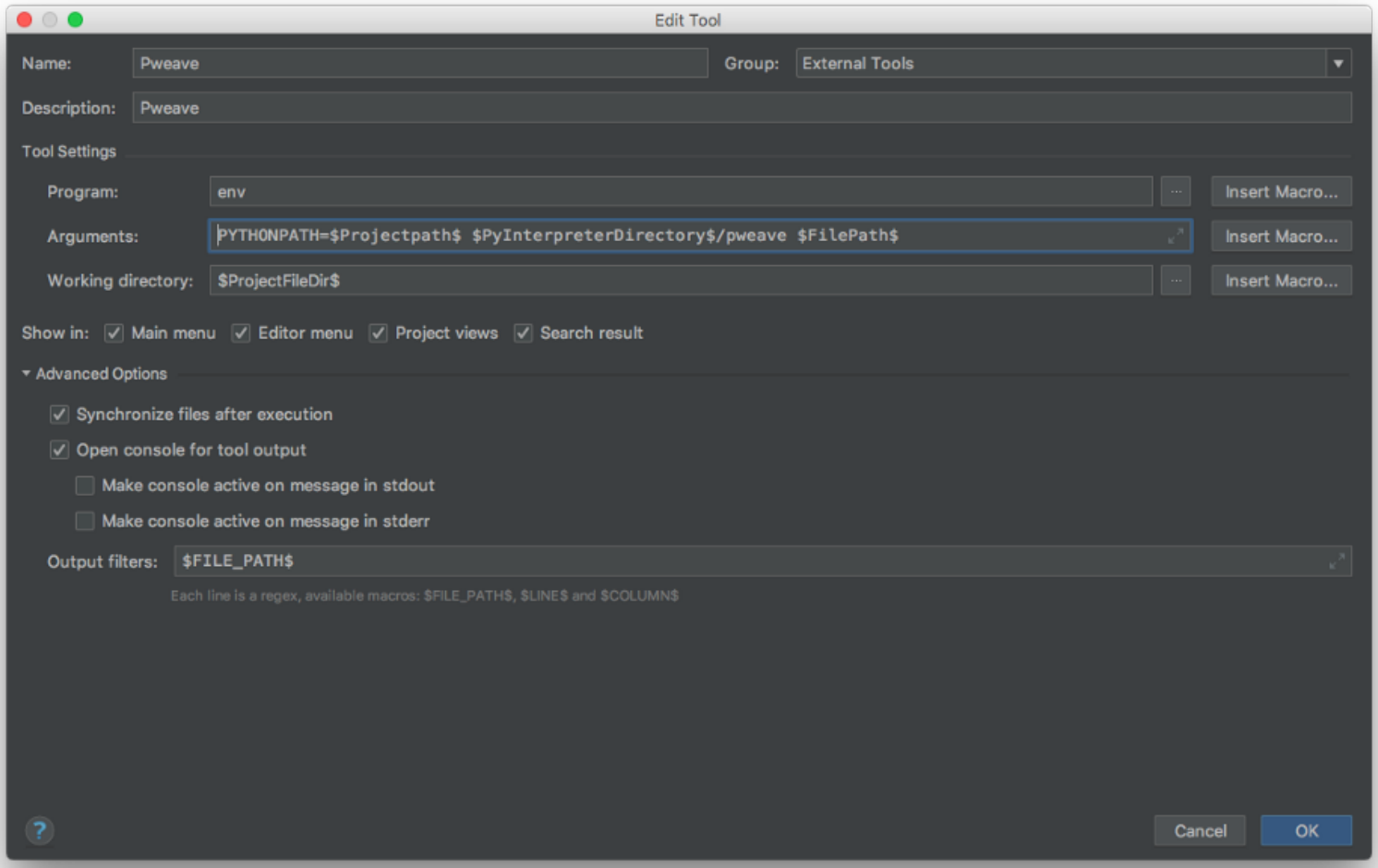
```
a = 2
```

a is now 2

Published from /Users/clementwalter/Library/Preferences/PyCharm2018.1/scratches/scratch_1.py using Pweave 0.30.3 on 14-02-2019.

I recommend setting up an External Tool in PyCharm to publish notebooks with a keyboard shortcut with the following configuration

(note the trick to add environment variables if necessary):



Pweave as an external tool configu ration

Conclusion

This is not another Why Jupyter notebooks suck article. I have nothing personal against this popular tool. I just wanted to share my personal experience using it. Especially working in a production driven environment, I have come to another workflow. What do you think about it?

. . .

Want more articles like this one? Don't forget to click the *follow* button here or on twitter.

Need help on your data

Our Agile teams are there for y

Email

Get in touch!